My Learning Journey

How To Make Games

In this article I'm going to be talking about the process behind actually making, testing and publishing a game. It's aimed at the absolute beginner but there should be something of interest here for the intermediate level reader as well. I'm going to be concentrating on making game with Adobe Flash, but the general techniques are applicable to any platform.

What is a game?

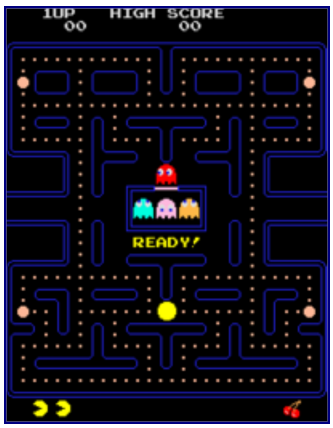
Wikipedia defines a game as:

"...structured playing, usually undertaken for enjoyment and sometimes used as an educational tool"

I would add that a game should generally contain these three components:

- Interactive element
- Challenge
- Goal

Obviously a game must be interactive, there must be some kind of challenge involved (mental or physical) and in most games there is a predefined goal which the player strives to achieve. For example, take Pac-Man, one of the most recognisable games there is:

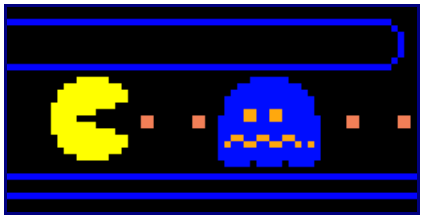


Pac-man

In Pac-man, the interactive component is that you control Pac-Man and can move him left/right/up/down. The goal is to eat all the pills and the challenge is that the ghosts chase Pac-Man and will kill him on touch. The way Pac-Man interacts with the maze-like environment also forms part of the challenge because the maze forces the player to choose the correct route ahead of time in the hopes that a ghost will not head him off at the pass, so to speak. Pac-Man also contains a nice twist which, in my view, is the key to the game's addictiveness; when you eat a large pill, all the ghosts turn blue and flee as Pac-Man is able to eat them on touch. This reversal of

the game rules provides an excellent contrast to the normal game-play; it feels extremely satisfying to eat a ghost and this reversal has the effect of balancing the game-play so it feels fair.

See this essay for a good description of fairness and it's importance in games.



Pac-Man about to eat a ghost

Within Pac-Man there are lessons in not only what a game is, but also what a great game is: that notion of fairness, the satisfaction of eating a ghost, the pleasure of a good job well done once all the pills are cleaned up.

This cleaning and satisfaction mechanic was explored again recently in a game called Katamari Damacy which is excellent fun.

The components of a game

Before we can understand how to make a game, we need to understand what components make up a game at it's most basic level.

- The rules of the game
- The graphics
- The sound

The rules of the game

Most important of all are the rules of the game. These define how the game functions. Here are some of the high-level rules for Pac-Man:

- When Pac-Man touches a pill, he collects it and earns a point
- Pac-Man must collect all pills in a level in order to progress to the next level
- When Pac-Man touches a ghost he loses a life, unless Pac-Man has eaten a large pill in which case the ghost gets eaten
- Ghosts pursue Pac-Man unless he has eaten a large pill, in which case they flee from him
- Walls cannot be passed through, and cause Pac-Man to stop

And so on. Without rules, a game simply ceases to be a game; with no interactive component and nothing to

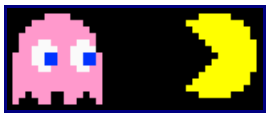
hold the game together it would just be a bunch of graphics and sounds randomly occuring.

The graphics

Second most important in a game are its graphics. In Pac-Man these are composed of two distinct types:

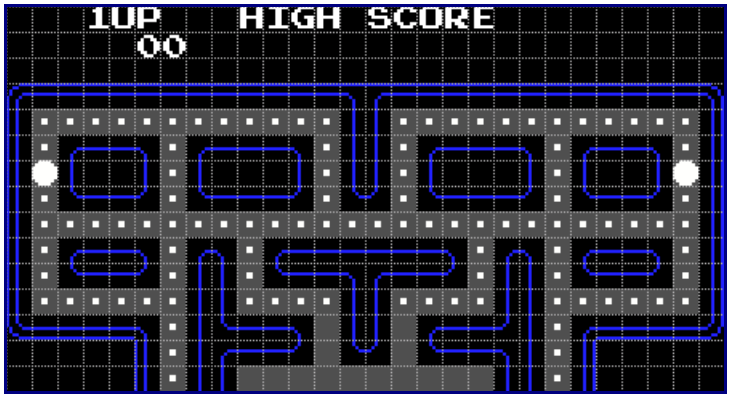
- Sprites
- Tiles

Sprites consist of anything which can move, like Pac-Man himself and the ghosts:



Sprites

The **tiles** makes up the rest of the graphics, and can be thought of as being very much like a font (as in a set of typographic characters), as each character is a fixed width and height (in this case 8x8 pixels):



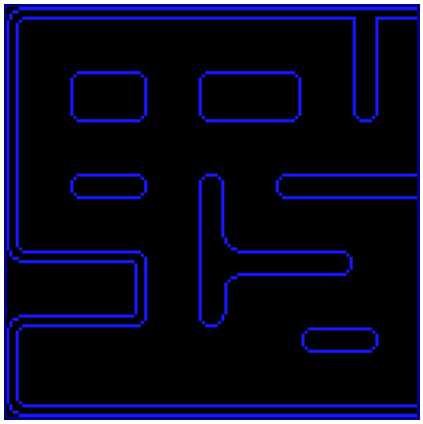
Tiles in Pac-Man

(image taken from the excellent The Pac-Man Dossier, which you can read for everything you ever wanted to know about Pac-Man)

Each little 8x8 square in the above image comes from the tile-set for Pac-Man, indeed you can see that this tile-set actually does contain the font as used by the game to display the score. If you took every unique tile in the above image and put them into a set, that would be all you needed to re-draw the Pac-Man world:

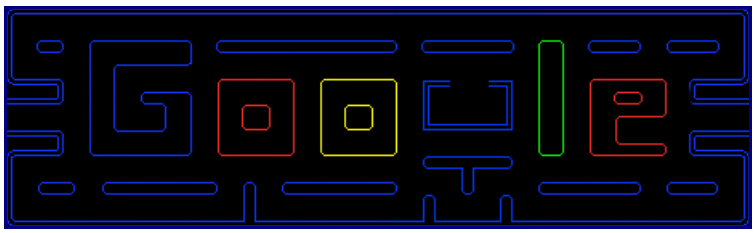
Reduced tile-set

The above represents a sub-set of what the full Pac-Man tile-set actually looks like. Taking it, I was able to reproduce a Pac-Man like level:



Level reconstructed from reduced tile-set

Notice that the everything in the level is composed solely of tiles from the reduced tile set? By creating a small set of building blocks which fit together like a jigsaw puzzle, it becomes possible to create an almost infinite variety of different levels. This process of reduction and re-composition is at the core of all games, even today. The Killzone series (PS2, PSP, PS3) for example makes heavy use of this technique, except it's now in 3d and they call their equivalent of tiles 'building blocks'.



Level from Google's 30th anniversary version

Above is a re-imagined Pac-Man level composed of the same tile-set which Google made for the 30th anniversary of Pac-Man. You might ask why game developers go to all this effort to reduce and then recreate what they already had in the first place. The reason is memory. It takes vastly less memory to recompose a level from a small set of building-blocks that it would without doing that; in the case of Pac-Man around 13 times less memory is taken by doing things this way.

Assuming each tile definition takes 8 bytes (8 bits x 8 bits), the Pac-Man level was 28x31 tiles - 28x31x8 = 6944 bytes to represent each tile individually. Compared to a tile-set of 32x2, 32x2x8 = 512 bytes. 6944/512 = 13.56.

Sound

The last major component of a game is sound, less important than the other two, but still vital to a game. Sound comes in the form of music and sound-effects in games. Take a listen to both in this clip: The intro music is instantly recognisable as are the sound effects. The in-game sound effects, particularly the siren like whirring noise, heighten the sense of urgency - once a large pill is eaten, the sound effects change to reinforce the game-play reversal which has occurred, making it even more compelling. As the game-play progresses and the level nears completion, the pitch of the siren noise in the background increases incrementally and with it the sense of urgency - this is great sound design; the sounds not only complement the game-play, they actually alter the player's perception of the game, making it feel much more frantic and thereby heightening the sense of fun.

Waveform from the SID chip (C64)

Back in the days of Pac-Man, all the sounds were programmed waveforms. The music was made up in much the same way as the level is composed from pieces of the tile-set; each particular instrument type would have been programmed for one particular note, and then these notes would have been sequenced to form the music, with just their pitch changing each time.

See this pdf article for more details on how early game sound designers engineered their work.

In modern times audio samples are used for sound effects and mp3s are used for the music. Sound effect samples are just tiny pieces of audio which are triggered to play in game when an event happens, such as when the player collects a power-up or loses a life.

How to make a game

Ok, so now we understand the component parts of a game, the question then becomes: how do we make our own? The answer to this question hinges on whether or not the reader has an understanding of programming. Either way, it's still possible to create a game, it's just that the path will be different.

Implementing the rules of a game

The way the most important component of a game, the rules, are implemented depends on whether the creator has any understanding of programming because these rules are often best expressed as a sequence of logical operations and conditions, which is very well suited to being implemented in a programming language. This doesn't mean that it's impossible to make a game without understanding programming, it just means the way in which one is constructed will be different.

The choice of target platform

Before we get into the nitty-gritty of how to actually go and make a game, it's important to consider the choice of the target-platform. For me, the chief driving force behind this choice is one of exposure; if I make a game, I want as many people as possible to actually get a chance to play it and give me feedback about what they liked/didn't like. Having a high exposure to your game can also present other opportunities like the chance to actually make some money from your creation.

Many choices

With this in mind, the target platform I'm going to be using in this article is Adobe Flash: it's installed on over 95% percent (source: riastats) of all computers connected to the internet, which is a massive number. It's a solid, mature platform with a lot of on-line knowledge and support available. Crucially, it also has a very large number of Portals such as Newgrounds and Kongregate which will take your game and host it for free, exposing it to millions of players. The cost of getting the same amount of exposure that a portal can provide for free, even for the best game would be prohibitive otherwise; this is currently one of the limiting factors for HTML5, as portals have yet to start supporting it in any great numbers. Note that you don't actually need to buy Adobe Flash in order to make games for that platform; there are freely available tools that let you build games which target the virtual machine which actually runs the games - in this case it's the Flash plugin in the browser you're reading this webpage with.

Technical fundamentals

At this point it's worth briefly covering some of the fundamentals of how a game works. The most important thing in the rendering (i.e. drawing the graphics) of a game are frames. Frames can be thought of like the individual images in a roll of cinema film. Keeping the same analogy, the monitor or TV which the game is being displayed on can be thought of as being both the projector and screen. The game is responsible for drawing each frame quick enough that monitor doesn't run out of film, so to speak - this is evidenced in some games where you get a visible lag occurring in a busy part of the game. In general, game developers strive to avoid this lagging as it breaks the player's immersion. Because of this need to draw every frame of the game, games are often synchronised with the monitor so that the game knows exactly when to run its logic and to draw the frame.

Monitors generally update at 60Hz or 60 times per second, but games are mostly 30Hz or 30 times per second, unless its a game which requires very fast response to player input, like a racing or fighting game.

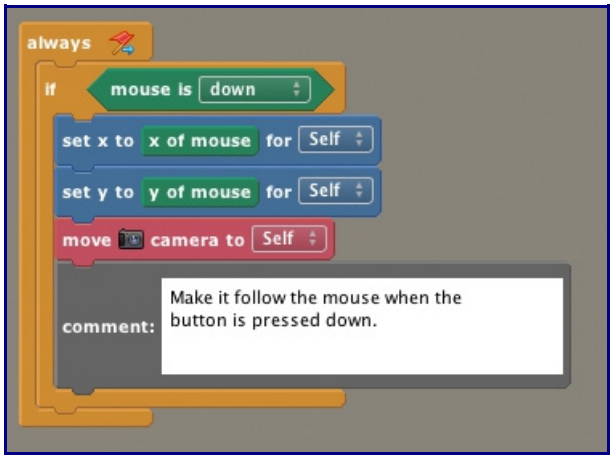
The baulk of the game logic (reactions to player inputs, collision detection etc) is therefore done once per frame. **Player input** is what drives a game; nearly everything which the game logic does is in response to player input in one way or another. Input can come in the form of mouse movements or key-presses. Typically, the main character in a game is driven by such inputs. Because the tactile experience of playing a game comes directly through the inputs, its very important that the response to these inputs feel 'right'. This can take a lot of experimentation and tweaking but the results are always worth it. **Collision Detection** is the process of detecting a collision between objects. In Pac-Man this is what stops Pac-Man from being able to pass through the walls and also enables the ghosts to catch Pac-Man when they touch him. **AI or Artificial Intelligence** is the thing which drives non player creatures, like the ghosts in Pac-Man. It doesn't need to be as complex as the name implies, just some simple rules will suffice, such as: move towards the player at a constant speed. The **Camera** is much like a movie camera in that it governs what we see on screen. Its needed because often the game world is larger than the screen (although not in the case of Pac-Man) and if they player goes outside the bounds of the screen, it would be useful if the camera moved so we can still see the player. It too has simple rules like: only show things which are within the game world and follow the player. With these fundamentals covered its time to get stuck in.

I don't know how to program

Not a problem. The tool we're going to be using to make games for Adobe Flash is called StencylWorks. Download this and have a read through their website to inspire yourself, it's free. The thing which makes StencylWorks so appealing is that it works in exactly the same way as I've been describing in this article.

StencylWorks

It works using a tile-set and actors (similar to Sprites), has a built-in level editor to allow you to place down the tiles to create your own levels and even has a solution in place for laying down the logical rules of the game without programming:

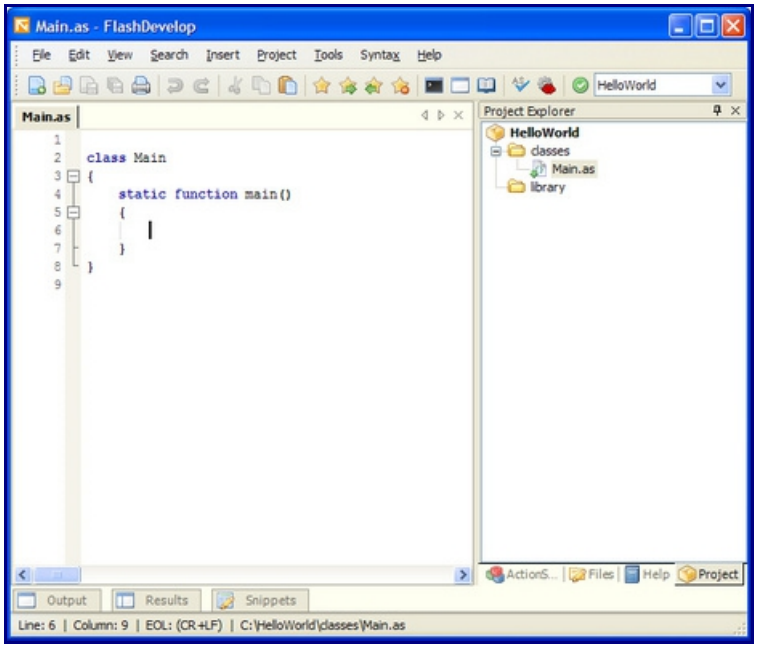


Drag and drop logic designer

The best way to get familiar with StencylWorks is to run through their interactive tutorial which should give you a nice knowledge base to build on - after following this simple tutorial you will have a player character moving around on screen and some enemies to avoid: Abigale's interactive stencyl tutorial (part 1) Here is another nice walk-through which covers making a Nyan Cat shoot-em-up game. On this page are a ton of video tutorials for StencylWorks which should keep you busy for a while, just scroll down to see them all. Once you have a basic understanding of the mechanisms requried to create your own game, it's time to start thinking about what kind of game you want to make, and how to make it good. Skip the I know how to program section to find out.

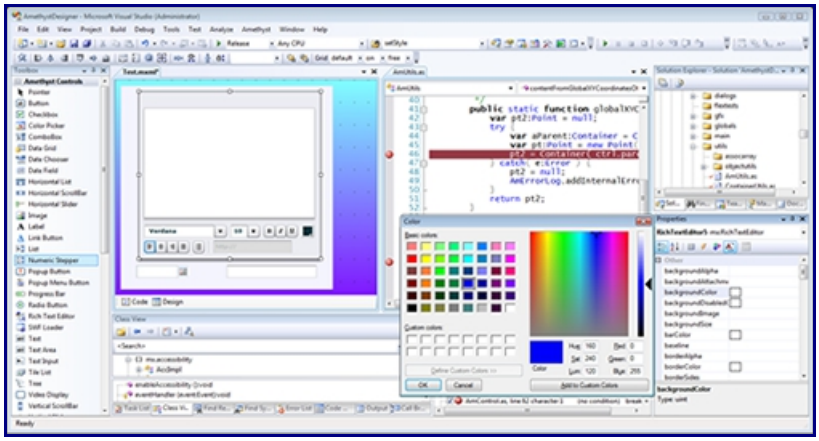
I know how to program

Ok, great! You can either choose to use StencylWorks as well, or you might find it more familiar to do all the logic with programming. To get set up, you'll want to download the Adobe Flex Compiler SDK and then either Flash Develop or Amethyst for Visual Studio, which is what I use. You'll need to set Flash Develop up to target the Flex SDK - some excellent documentation to help you out can be found: Installing Flash Develop Configuring Flash Develop



Flash Develop

Some instructions for installing Amethyst can be found here. Note that Amethyst requires you have Visual Studio already, although you can download a shell version of visual studio 2010 for free which you can then install the 60 day trial of Amethyst on top of. The choice of which to use depends on whether or not you are already familiar with Visual Studio. If you are, I strongly recommend Amethyst because it lets you take advantage of all the features of VS, including the excellent debugger.



Amethyst

Once you're up and running it's time to get stuck in with learning how Flash works. Here is the ActionScript 3.0 reference manual from Adobe which is essential reading for when you want to find out what a particular function or attribute does. Here is an excellent video on getting started with FlashDevelop. Here is a big list of

reference material and tutorials which should help you get started. Now that you've understood the basics and have stuff moving around on screen, it's time to start learning some more complex stuff. Here are some of the articles which can be found on this site and are specifically designed for this level of skill: How to make a 2d platform game How to make a 2d platform game - collision detection How to make a 2d platform game - ladders and ai A more advanced tutorial can be found here: How to make Angry Birds - part 1 How to make Angry Birds - part 2

How do you decide what type of game to make?

The key to this choice is to keep it really **small and simple**. The truth of the matter is that games are hard work to produce and take a lot of time and effort; the number one thing which will stop you from releasing a game is simply not finishing it. **This cannot be emphasised enough**. Take a look at some successful, yet small games such as Winterbells or Totem Destroyer to get a rough idea of the size of the game you should be thinking about.

Winterbells and Totem Destroyer

Once you start building up a body of completed games you will find it much easier to complete larger projects because you'll be able to draw on your earlier work to help you. Winterbells is a good example of compact game design; the game only has one game mechanic but it does it well enough to make the game addictive. When you're starting out making games it's a good idea to follow the same pattern: just pick one game mechanic and implement it very well. **Well enough that you could make the whole game just about that.** Once you get the hang of that, you can then start to introduce multiple mechanics which work together to form a genre game, such as a shoot-em-up or platformer. When starting out, it's best to not even consider trying to make money from your first few games; instead, treat them as an investment in experience.

Here is an excellent article about how game mechanics can be used to build good games.

How do you come up with a game idea?

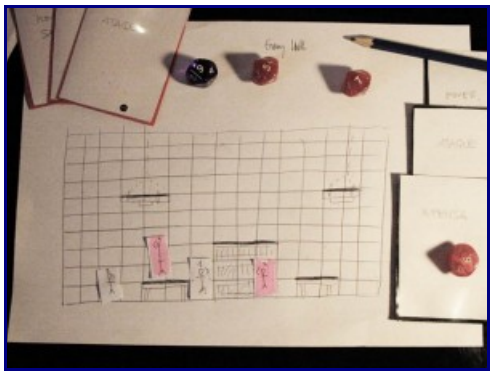
Take some of your favourite games, look over at the game mechanics they contain (i.e. the mechanisms which enable the game-play, an example would be eating a large pill in Pac-Man and being able to chase the ghosts), pick some that you really like and see if you can put a unique twist on them. An example would be something like this: To get inspiration spend some time playing old arcade games in Mame; there are an absolute ton of good ideas just waiting to be re-awakened. Resist the temptation to copy an existing game, you should instead strive to provide something new and fresh. The reason for this is that when you copy an existing game you start to lose that creative mindset which is so important and will tend to keep falling back on doing things

exactly how the game you are copying did them. Avoid this at all costs. It's fine to use concepts from other games, but you must be free creatively.

Make a prototype

Once you've got your game idea, you should try to construct a prototype to test it out. If possible it's a good idea to prototype the rough ideas for a game on paper first before even attempting to make a playable version. Keep it simple, though.

Here is an interesting walk through of a paper prototype for one partciular game.



An on-paper prototype

Once you have the basic paper prototype done its time to move on to the playable prototype. This should be as simple as possible; start out with the core mechanic of the game. Don't worry about things like intro-screens or buttons or other paraphernalia, just concentrate on the actual game-play itself; you probably wont even need sound at this point, but make sure there is a challenge and there is a goal of some kind.

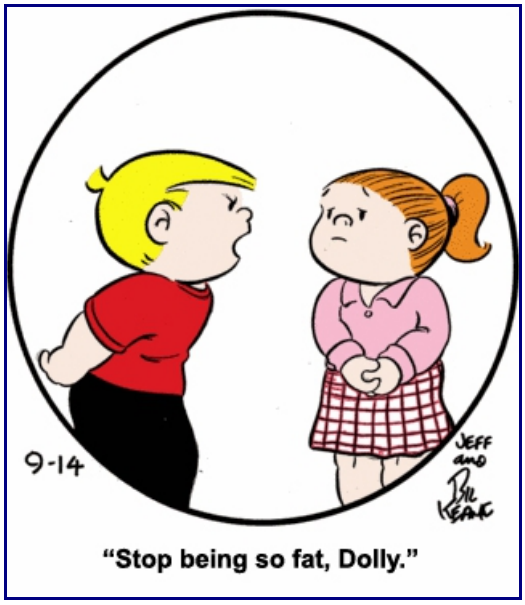
Side note: I was writing a simple 2d platform game for one of my tutorials just recently when I noticed that the game wasn't really fun. The reason was that I didn't have any concept of lives or energy (because I hadn't implemented them yet). Once they were implemented, the game suddenly took on a new angle; the mere fact that you could die and get sent back to the start of the level made you care about the progress you were making through the level, thereby creating that missing element of fun. My problem was that I'd missed out the **Challenge** requirement from what makes a game.

The idea of the prototype is that it lets you try out the core of the game in the smallest amount of time. It's very important that this core element be fun and engaging otherwise it's highly unlikely that the rest of the game will be any good. Once the prototype is complete and is fun, you can start to think about how the rest of the game fits around it. It's recommended to throw the prototype project away and start again from scratch, keeping the design and assets. The reason for this is that the prototype is designed to be thrown together very quickly, which is generally not something you'd want to build a full blown game on top of. By starting again you will be able to fix all the little problems you noticed when building the prototype, since everyone performs a complex task better the second time around. This technique is used by all the major game companies today.

See this article for more details on prototyping. Here is another detailing how the on-paper prototype fits with the rest of the design process

How do you make a good game?

This is the most difficult part of all. You've got your game idea, you've made a prototype, but how do you know if it's any good? Once you've been working on a game for a while it does get rather difficult to tell if what you've been making is fun. The reason is that being constantly exposed to every minute detail of the game it becomes very difficult to see the wood for the trees. A good way to tell is to leave the game overnight and if you find yourself thinking about wanting to play it the next day, you're probably on to something. The next thing to do is to get **as many people as possible** to play it and let you know what they think. It's better if the people aren't family members because you'll tend to get a rose-tinted spectacles response that way. Collect all their feedback and try to remember one thing: **every single piece of constructive negative feedback is correct**. Everyone who takes time out to tell you want they don't like is right.



Negative feedback

This is a very difficult thing to take on board and it's so, so easy to just start dismissing individual pieces of feedback because they're inconvenient to fix or would take too long. But you must address these issues because if you don't you will end up with a poorer game in the end at which point it will be too late to fix them. Believe me there is nothing worse than seeing a negative review online which mentions one of those bits of feedback that you ignored. Keep a tally of the most common issues and sort them by frequency; addressing problems in this way allows you to concentrate on fixing the main issues with the game. If you get a lot of feedback that the game isn't fun, stop work on it right away and come up with a new game idea and a new prototype. It will seem like a waste, but it's an essential step towards becoming a good game designer. In fact it's probably best to come up with a number of potential game ideas at the initial design phase so you have some other work to fall back on. Starting over is never really starting from scratch in game creation because you'll be able to save some parts of the work you're discarding, whether that be the graphics, or some logic or whatever, it's all part of the creative process.

Here is another article on the role of user feedback in game design

Bug-testing

Once your game is complete, you'll want to start getting as many people as possible to play it in order to find

all the little bugs that you've missed.

No matter how well you think you've tested it yourself, you can guarantee that the minute you hand it over to someone else they'll find a bunch of horrible problems that you've missed.

A good tip for the intermediate/advanced level programmer is to install an exception handler to catch all unhandled exceptions and display a message on screen (with a call-stack) so the person playing knows they found a bug and should report it to you. Remember to make the message text selectable so the player can copy/paste it. Even better, if you have a server with a database configured, submit this message directly into the database.

After you've done this it's time to move into testing stage 2, whereby you should sign-up to http://www.flashgamelicense.com as a developer and submit your game for evaluation and testing by other developers only. That means only other developers can see your game and they should be able to provide you with more feedback on any bugs and also give you a review. Once you've done this, gotten as much feedback as possible and fixed any last bugs it's time to think about releasing your game into the wild!

Publishing your game

Congratulations! You're now ready to show your game to the world! Doing this is usually as simple as creating an account at the major portals and uploading your game.

Here is a list of the major portals ranked by website popularity.

Looking through that list you might start to get the impression that you've got your work cut out for you. However, once you've spot picked a few of your favourite portals, you can then upload your game to http://flashgamedistribution.com/ which will do a lot of the work for you via a method called viral distribution. Viral distribution is the process whereby portal owners will see a game they like the look of and take it for use on their own portal, thereby spreading it around the internet. They know that http://flashgamedistribution.com/ has been specially designed to assist in this process so a lot of portal owners go there on a regular basis. To be honest, they also go to the major portals as well, so this process may well just happen automatically.

Good luck!

I've covered every stage of game development from concept to implementation through testing and release. Making a game is not an easy process as you've probably gathered by now; it's much easier to make a few simple game mechanic prototypes though, so don't be put off just yet. The best advice I can give you is to be persistent and never give up; much like anything in life, making a game requires a huge amount of perseverance especially if you're just starting out. Take things step by step and treat every bit of progress as a triumph. Until next time, have fun! Cheers, Paul Source http://www.wildbunny.co.uk/blog/2012/01/06/how-to-make-games/